```
!pip install transformers rouge-score sacrebleu bert-score tensorflow

import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold
from transformers import T5Tokenizer, T5ForConditionalGeneration, BartTokenizer, BartForConditionalGeneration, PegasusTokenizer, PegasusForCo
from rouge_score import rouge_scorer
import sacrebleu
from bert_score import score as bert_score
import tensorflow as tf
```

Show hidden output

```
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")



# Define the models and tokenizers
model_names = {
    "T5-Small": "t5-small",
    "T5-Base": "t5-base",
    "BART": "facebook/bart-large-cnn",
    "Pegasus": "google/pegasus-xsum",
    "MT5-Base": "google/mt5-base"
}

# Initialize models and tokenizers
models = {}
tokenizers = {}

for name, model_id in model_names.items():
    if "T5" in name or "MT5" in name:
        tokenizer = T5Tokenizer.from_pretrained(model_id)
        model = T5ForConditionalGeneration.from_pretrained(model_id)
    elif "BART" in name:
        tokenizer = BartTokenizer.from_pretrained(model_id)
        model = BartForConditionalGeneration.from_pretrained(model_id)
    elif "Pegasus" in name:
        tokenizer = PegasusTokenizer.from_pretrained(model_id)
        model = PegasusForConditionalGeneration.from_pretrained(model_id)
    models[name] = model
    tokenizers[name] = tokenizer
```

Show hidden output

```
import warnings
warnings.filterwarnings("ignore")



from sklearn.model_selection import KFold
from nltk.translate.bleu_score import corpus_bleu




# Load dataset (example dataset)
data = pd.DataFrame({
    'input_text': [
        """Social media are interactive technologies that facilitate the creation, sharing, and aggregation of content (such as ideas, intere

        Online platforms that enable users to create and share content and participate in social networking.
        User-generated content—such as text posts or comments, digital photos or videos, and data generated through online interactions.
        Service-specific profiles that are designed and maintained by the social media organization.
        Social media helps the of online social networks by connecting a user's profile with those of other individuals or groups.
        The term "social" in regard to media suggests platforms enable communal activity. Social media can enhance and extend human networks.

        Popular social media platforms with over 100 million registered users include Twitter, Facebook, WeChat, ShareChat, Instagram, Pinter
```

```
        Social media outlets differ from old media (e.g. newspapers, TV, and radio broadcasting) in many ways, including quality, reach, freq

        Social media has been criticized for a range of negative impacts on children and teenagers, including exposure to inappropriate conte
        "A social network is a social structure consisting of a set of social actors (such as individuals or organizations), sets of dyadic t
    ],
    'reference_summary': [
        "Social media are interactive platforms that enable users to create, share, and engage with content within virtual communities. These
        "A social network is a structure of interconnected individuals or organizations linked by relationships and interactions. Social netw
    ]
})

# Define the number of folds
k = 2
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Function to summarize text
def summarize_text(model, tokenizer, input_text):
    inputs = tokenizer.encode("summarize: " + input_text, return_tensors="pt", max_length=512, truncation=True)
    summary_ids = model.generate(inputs, max_length=150, min_length=30, length_penalty=2.0, num_beams=4, early_stopping=True)
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    return summary

# Function to evaluate summaries
def evaluate_summary(summary, reference_summary):
    # ROUGE
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'])
    rouge_scores = scorer.score(reference_summary, summary)

    # BLEU
    bleu_score = sacrebleu.corpus_bleu([summary], [[reference_summary]]).score


    # BERTScore
    P, R, F1 = bert_score([summary], [reference_summary], lang="en")

    return {
        "ROUGE1": rouge_scores['rouge1'].fmeasure,
        "ROUGE2": rouge_scores['rouge2'].fmeasure,
        "ROUGEL": rouge_scores['rougeL'].fmeasure,
        "BLEU": bleu_score,
        "BERTScore": F1.item()
    }

# Perform k-fold cross-validation
results = {name: [] for name in model_names}

for train_index, test_index in kf.split(data):
    train_data = data.iloc[train_index]
    test_data = data.iloc[test_index]

    for name, model in models.items():
        for _, row in test_data.iterrows():
            start_time = time.time()
            summary = summarize_text(model, tokenizers[name], row['input_text'])
            end_time = time.time()
            execution_time = end_time - start_time
            evaluation = evaluate_summary(summary, row['reference_summary'])

            # Print the generated summary
            print(f"Model: {name}")
            print(f"Input Text: {row['input_text']}")
            print(f"Reference Summary: {row['reference_summary']}")
            print(f"Generated Summary: {summary}")
            print("\n")

            results[name].append({
                "Input Text": row['input_text'],
                "Reference Summary": row['reference_summary'],
                "Generated Summary": summary,
                "Execution Time": execution_time,
                "Evaluation": evaluation
            })

# Convert results to DataFrame
results_df = {name: pd.DataFrame(results[name]) for name in model_names}
```

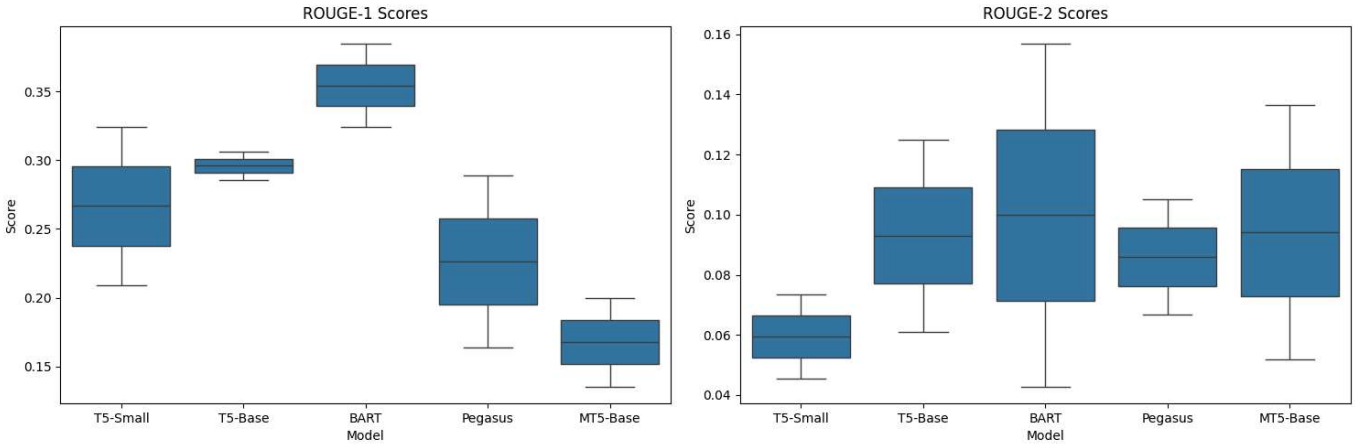
Show hidden output

```
def error_analysis(df):
    df['ROUGE1_diff'] = df['Evaluation'].apply(lambda x: x['ROUGE1'])
    df['ROUGE2_diff'] = df['Evaluation'].apply(lambda x: x['ROUGE2'])
    df['ROUGEL_diff'] = df['Evaluation'].apply(lambda x: x['ROUGEL'])
    df['BLEU_diff'] = df['Evaluation'].apply(lambda x: x['BLEU'])
    df['BERTScore_diff'] = df['Evaluation'].apply(lambda x: x['BERTScore'])


    worst_case = df.loc[df['ROUGE1_diff'].idxmin()]
    best_case = df.loc[df['ROUGE1_diff'].idxmax()]

    return worst_case, best_case


error_analysis_results = {name: error_analysis(results_df[name]) for name in model_names}


for name, (worst_case, best_case) in error_analysis_results.items():
    print(f"Model: {name}")
    print("Worst Case:")
    print(worst_case)
    print("Best Case:")
    print(best_case)
    print("\n")
```

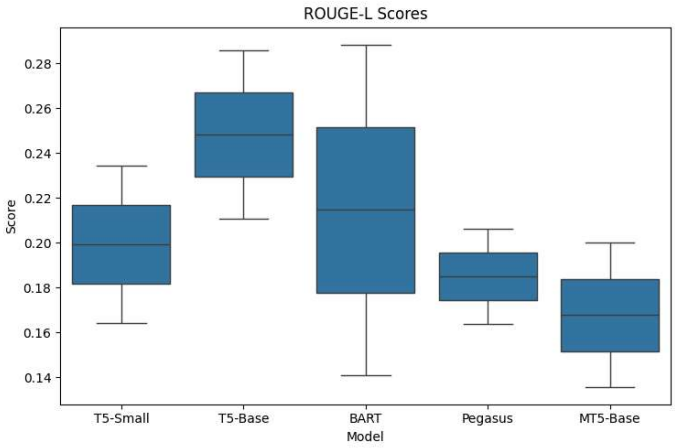
Show hidden output

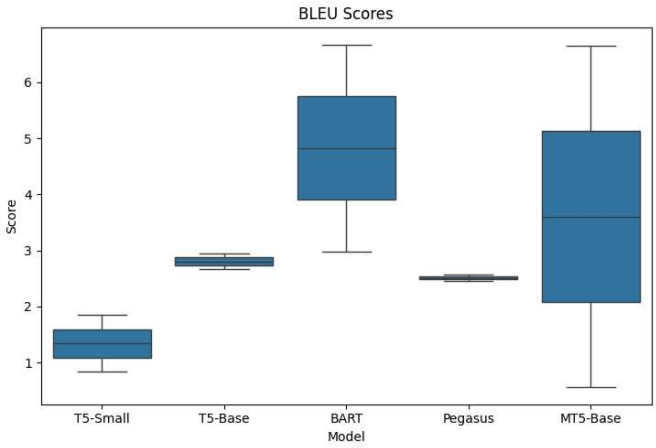
```
def visualize_attention(model, tokenizer, input_text):
    inputs = tokenizer.encode("summarize: " + input_text, return_tensors="pt", max_length=512, truncation=True)

    outputs = model.generate(inputs, max_length=150, min_length=30, length_penalty=2.0, num_beams=4, early_stopping=True, return_dict_in_gen


    attentions = outputs.attentions

    if attentions is None:
        print("Attention weights are not available for this model during generation.")
        return

    attentions = attentions[-1]
    attentions = attentions[0].detach().numpy()

    # Plot attention
    fig, ax = plt.subplots(figsize=(10, 10))
    sns.heatmap(attentions.mean(axis=0), cmap='viridis', ax=ax)
    ax.set_title('Attention Weights')
    ax.set_xlabel('Input Tokens')
    ax.set_ylabel('Output Tokens')
    plt.show()



import seaborn as sns



# Convert evaluations to a DataFrame
evaluation_df = pd.concat([df[['ROUGE1_diff', 'ROUGE2_diff', 'ROUGEL_diff', 'BLEU_diff', 'BERTScore_diff']] for df in results_df.values()],

# Plot the results
fig, ax = plt.subplots(2, 2, figsize=(15, 10))

# ROUGE1
sns.boxplot(x='Model', y='ROUGE1_diff', data=evaluation_df, ax=ax[0, 0])
ax[0, 0].set_title('ROUGE-1 Scores')
ax[0, 0].set_ylabel('Score')

# ROUGE2
sns.boxplot(x='Model', y='ROUGE2_diff', data=evaluation_df, ax=ax[0, 1])
ax[0, 1].set_title('ROUGE-2 Scores')
ax[0, 1].set_ylabel('Score')

# ROUGEL
```

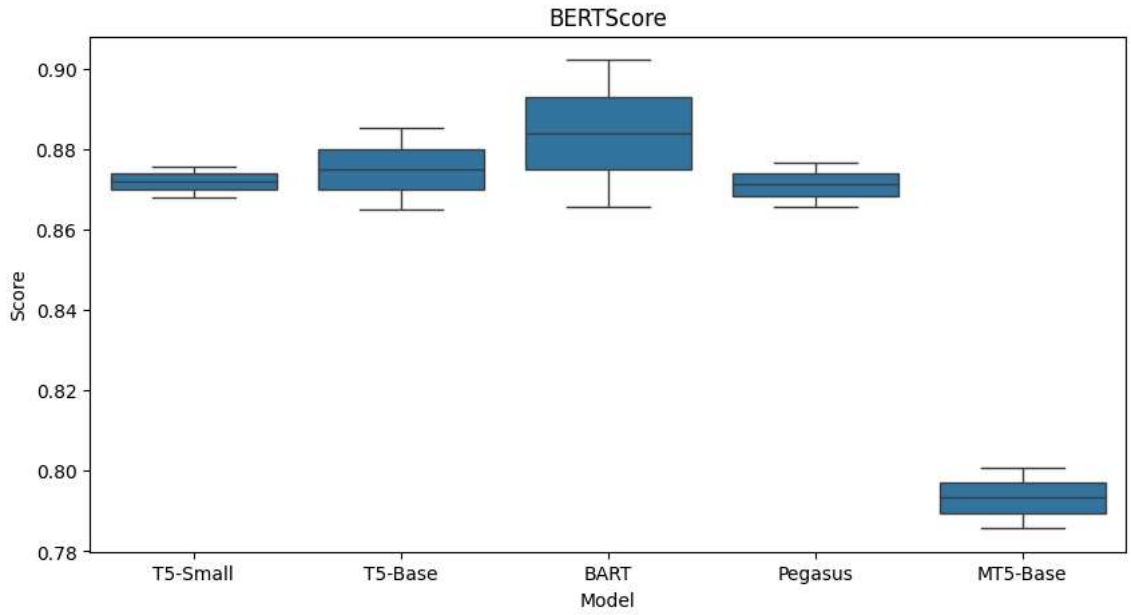
```
sns.boxplot(x='Model', y='ROUGEL_diff', data=evaluation_df, ax=ax[1, 0])
ax[1, 0].set_title('ROUGE-L Scores')
ax[1, 0].set_ylabel('Score')

# BLEU
sns.boxplot(x='Model', y='BLEU_diff', data=evaluation_df, ax=ax[1, 1])
ax[1, 1].set_title('BLEU Scores')
ax[1, 1].set_ylabel('Score')

plt.tight_layout()
plt.show()

# Plot BERTScore
plt.figure(figsize=(10, 5))
sns.boxplot(x='Model', y='BERTScore_diff', data=evaluation_df)
plt.title('BERTScore')
plt.ylabel('Score')
plt.show()

# Plot execution times
execution_times = pd.concat([df[['Execution Time']] for df in results_df.values()], keys=results_df.keys()).reset_index(level=0).rename(colu
plt.figure(figsize=(10, 5))
sns.boxplot(x='Model', y='Execution Time', data=execution_times)
plt.title('Execution Times')
plt.ylabel('Time (seconds)')
plt.show()
```


ROUGE-1 Scores
Score
0.35
0.30
0.25
0.20
0.15
T5-Small
T5-Base
BART
Pegasus
MT5-Base
Model

ROUGE-2 Scores
Score
0.16
0.14
0.12
0.10
0.08
0.06
0.04
T5-Small
T5-Base
BART
Pegasus
MT5-Base
Model


ROUGE-L Scores
Score
0.28
0.26
0.24
0.22
0.20
0.18
0.16
0.14
T5-Small
T5-Base
BART
Pegasus
MT5-Base
Model


BLEU Scores
Score
6
5
4
3
2
1
T5-Small
T5-Base
BART
Pegasus
MT5-Base
Model


BERTScore
Score
0.90
0.88
0.86
0.84
0.82
0.80
0.78
T5-Small
T5-Base
BART
Pegasus
MT5-Base
Model